Resource Control and Reservation

- policing: hold sources to committed resources
- scheduling: isolate flows, guarantees
- resource reservation: establish flows
Usage parameter control: leaky bucket algorithm

- constrain what host can inject into the network
- single server queue with fixed service time
- finite-size bucket ⇒ either throttle source or loose packets
- no burstiness allowed
Token bucket

- *tokens* allow bursts into the network
- tokens generated at constant rate up to maximum burst size
- if no token, either quench source or drop packet
- implementation: token counter, incremented periodically

Generic Cell Rate Algorithm (GCRA)
Mechanism used by UNI 3.1 to police either peak or mean cell rate.

**PCR:** peak cell rate

**SCR:** sustainable cell rate = mean cell rate

**CDVT:** cell delay variation tolerance

\[ \tau_s : \text{burst tolerance} \]

\[
\begin{array}{ccc}
\text{peak rate} & \text{mean rate} \\
T & 1/\text{PCR} & 1/\text{SCR} \\
L & \text{CDVT} & \tau_s
\end{array}
\]
• cell $i$ can arrive at $t_i > t_{i-1} + T - L$; but: arrival time set to $t_i = t_{i-1} + T$

• can’t save up late arrivals

• can’t accumulate $L$

GCRA flow chart

TAT = theoretical arrival time
GCRA

Packet scheduling

**work conserving:** never delay a packet if line is idle ➞ no lower bound on jitter

**non-work-conserving:** minimum residency time ➞ jitter bound

*Isolation:* one misbehaving source can’t monopolize resources
FIFO+ and HL

For packets with real-time constraints (deadlines) give priority to those about to miss their deadline

**hop-laxity**: priority = \( \frac{\text{hops to go}}{\text{time left}} \)

don't drop packets that have exceeded their deadline or are too close

**FIFO+**: give priority to packets if travel time > average for class

- both require accumulating delays
- performance better than FIFO
- but: no guarantees, scheduling overhead

---

**Weighted Fair Queueing (WFQ)**

- fair queueing: separate queues for each input stream, round-robin favors long packets, wait for \( n \) other queues if a bit too late

- WFQ: order transmissions by when last bit would have been sent under bit-by-bit round robin

- need ordered queue of size \( q \): \( O(\log q) \) expensive

- divide bandwidth into \( m \)-bit cycles and distribute unequally
Weighted Fair Queueing

Delay $D_i$ of flow $i$ if token bucket at edge:

$$D_i = \frac{\beta_i}{g_i} + \frac{(h_i - 1)l_i}{g_i} + \sum_{m=1}^{h_i} \frac{l_m}{r_m}$$

where $\beta$: bucket size; $g_i$: fraction; $l_i$: maximum packet length for $i$; $l_\star$: maximum packet length in network; $h_i$: number of hops; $r_m$: outbound bandwidth

Reservations

First approach: everybody is the same ➞ best effort ➞

- enough bandwidth for everybody (telephone network)
- “human backoff” if unusable
- TCP for data applications (but: also minimum usable bandwidth)
- adjust audio or video coding to best possible ➞ application control (later)
- pick least congested route: telephone system, but Internet too large
Reservations

Some are more equal than others ➞

- incumbency protection
- priorities (general over PFC)
- bulk service vs. priority delivery ➞ cost

Reservations

$/kb/s may be dynamic ➞

- reservation may change during the lifetime of an application
- networks may not be homogeneous ➞ different multicast groups for different layers or versions
RSVP

Receiver-oriented, out-of-band reservation protocol standardized by IETF:

- not a routing protocol, but interacts with routing
- may need QOS routing to pick appropriate path
- transports opaque QOS and policy parameters for sessions
- flow: group of packets being treated the same ➞ same multicast group or destination, IPv6 flow id, ...
- simplex ➞ setup for unidirectional data flows

RSVP, cont’d.

- does not prescribe admission or policy control
- sets up packet classifier, but does not handle packets
- independent sessions (can’t tie video and audio session)
- multicast (and unicast)
- either own protocol type or UDP encapsulated
RSVP Objects

Flow descriptor =

**Flowspec:**
- service class
  - Rspec $\Rightarrow$ desired QoS
  - Tspec $\Rightarrow$ describes traffic characteristics

**Filterspec:** which packets get this treatment $\Rightarrow$ sender IP address/port, protocol, other fields $\Rightarrow$ complex (regular expressions? IP options!) $\Rightarrow$ currently, sender IP address and UDP/TCP port $\Rightarrow$ no fragmentation

Reservation Styles

<table>
<thead>
<tr>
<th>sender selection</th>
<th>reservations</th>
</tr>
</thead>
<tbody>
<tr>
<td>explicit</td>
<td>fixed filter (FF)</td>
</tr>
<tr>
<td>wildcard (all)</td>
<td>shared-explicit (SE)</td>
</tr>
<tr>
<td></td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>wildcard filter (WF)</td>
</tr>
</tbody>
</table>

$\Rightarrow$ mutually incompatible

**explicit:** list senders by address

**wildcard:** any sender with a specific port (e.g.)

**shared:** only one active data source $\Rightarrow$ e.g., reserve for twice needed for audio

**distinct:** video
**RSVP: basic operation**

- receiver joins group via IGMP
- source sends PATH messages to receivers ➞ same path as data: previous hop to source, Tspec ↔ RESV one path, data another
- receivers send RESV messages back to senders

**RSVP: basic operation**

- reservations may be lowered
- reservations are merged at each node for same sender: max. flowspec
- merge point or data sender may send confirmation (if requested)
- reservations *may* get merged between senders (audio!)
- one-pass ➞ receiver doesn’t know final QoS ➞ One Pass With Advertising
- application *should* explicitly tear down reservations
Killer Reservations

1. small reservation in place; another receiver larger reservation ➔ failure? ➔ keep old

2. large reservation fails again and again ➔ blocks new, smaller one

RSVP service classes

guaranteed: no loss, upper bound on delay

controlled load: “few” losses, “like unloaded network” ➔ delay-adaptive applications

best effort: no guarantees; current IP service model ➔ delay + bandwidth adaptive services

others: research
## RSVP vs. ATM resource reservation

<table>
<thead>
<tr>
<th>multicast tree, reservation</th>
<th>IP, RSVP</th>
<th>ATM</th>
</tr>
</thead>
<tbody>
<tr>
<td>origin</td>
<td>receiver</td>
<td>sender (root) ➞ UNI4.0</td>
</tr>
<tr>
<td>change reservations</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>routing changes</td>
<td>time-out</td>
<td>re-establish VC</td>
</tr>
<tr>
<td>routing</td>
<td>IP routing</td>
<td>PNNI (QOS)</td>
</tr>
<tr>
<td>flow merging (audio)</td>
<td>yes</td>
<td>no (separate VCs)</td>
</tr>
<tr>
<td>receiver diversity</td>
<td>not yet</td>
<td>no</td>
</tr>
<tr>
<td>state</td>
<td>soft</td>
<td>hard</td>
</tr>
</tbody>
</table>

## The recurring costs of reservations

**Signaling:** processing and state maintenance, APIs

**Routing:** QoS path selection, state distribution

**Policy:** who gets what (and who doesn’t)

**Charging, billing, accounting, service contracts:** right party pays for usage, ensure QoS is delivered as promised
RSVP implementation

- scheduling: about 10% cost overhead
- low-end 68040: 0.73 ms for PATH, 0.37 ms for RESV
- approximately 1,000 flow setups/s
- processing of PATH (RESV) refresh: 0.33 ms (0.29 ms)
- approximate capacity is 1,600 flows
- about 500 bytes/flow
- refresh bandwidth ≈ 100 kb/s for 1000 flows (30 s refresh)
- PATH: 208 bytes, RESV: 148 bytes

Resource reservation: general comments

- doesn’t help if network capacity ≪ demand
- modes:
  - receiver-oriented: RSVP
  - sender-oriented: YESSIR
- scaling issues: a reservation for every phone call ↔ datagram idea, routing aggregation
RSVP problems

- if reservation/tear down request lost, no immediate feedback
- can increase reservation latency or “phone off hook”
- large number of refreshes ➔ scaling problems

➔ hop-by-hop confirmation (→ extend refresh interval)

RSVP scaling

Scaling issues:

- number of flow states ➔ refresh, memory, time-outs
- large number of packet queues

Alternatives:

- “tunnels” = encapsulation IP-in-IP ➔ overhead
- aggregation for sender reservation ➔ flow classes
- drop and delay preferences
**YESSIR**: Yet another Sender Session Internet Reservation

- RSVP: separate daemon, API
- Interegrate into application that needs it (embedded systems!)
- In-band ⇒ easier firewall
- Router alert option
- Soft-state + RTCP BYE
- Partial reservations: add links as session ages ↔ fragmentation

---

### YESSIR

Plain RTCP SRs or additional information:

<table>
<thead>
<tr>
<th>IP Header with Router-Alert Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP Header</td>
</tr>
<tr>
<td>RTCP message:</td>
</tr>
<tr>
<td>- Sender Report:</td>
</tr>
<tr>
<td>- sender information</td>
</tr>
<tr>
<td>- detailed report for each source</td>
</tr>
<tr>
<td>YESSIR message:</td>
</tr>
<tr>
<td>- reservation command: active/passive</td>
</tr>
<tr>
<td>- reservation style, refresh interval</td>
</tr>
<tr>
<td>- reservation flow specification</td>
</tr>
<tr>
<td>- link resource collection</td>
</tr>
<tr>
<td>- reservation failure report</td>
</tr>
<tr>
<td>Profile-specific extensions</td>
</tr>
</tbody>
</table>

End-to-end refresh (vs. hop-by-hop)
YESSIR

- measurement mode
- IntServ flow specs
- PT-based for well-known PTs
- TOS-based: value
- killer reservations ➞ SR reservation failure
- OPWA: hop count, propagation delay, aggregated bandwidth, delay bounds ➞ updated at router
- cost: 360 μs

SRP: Scalable Reservation Protocol

- sender-oriented, out-of-band
- data packets marked as REQUEST ➞ learn reservation level
- router aggregates requests, downgrades to best effort
- receiver reports rate of successful REQUESTS
- sender adjusts rate RESERVED data packets
- aggregation by estimation:
- max(observed traffic over several intervals)
- effective bandwidth $e = \sup_{t_j - t_i + D} \sum n_i$
SRP packet processing

SRP estimator

Reserved
Request
Best effort

Update the estimated bandwidth

Is an update of the estimated bandwidth acceptable?

Packet scheduler

Reserved
Request
Best effort

Can the packet be schedule in the reserved service class?

Yes
No

Can the packet be schedule in the best effort class?

Yes
No

Discard